Grundläggande programmering och datalogi 2012/2013

för F1 och SIMTEK

Matlab del 2

Mål

Kunskaperna du fått i första MATLAB-uppgiften befästs. Dessutom ska du efter den här laborationen med hjälp av MATLAB och minstakvadratmetoden kunna anpassa linjer, sinuskurvor och cirklar till mätdata.

Rekursion är en viktig teknik i datalogin. Vi övar på rekursion genom att rita ett rekursivt mönster.

Redovisning

Grunduppgifter, obligatoriska: 1, 2, 3 och 4.

Extrauppgifter, för betygshöjning: Gör uppgift 5 och **en** av uppgifterna 6–7.
En av de sista uppgifterna får alltså väljas bort.

Datafiler

Alla datafiler kan kopieras från kursbiblioteket eller från labbwebbsidan.
Kursbibliotekskatalogen för MATLAB är `/info/grupdat12/matlab/` och filerna heter

```
/info/grupdat12/matlab/data1.m
/info/grupdat12/matlab/vinter.mat
/info/grupdat12/matlab/data2.m
```

Uppgift 1 – Rät linje med minstakvadratmetoden

Använd minstakvadratmetoden för att anpassa en rät linje $y = c_1 + c_2 x$ till de mätdata som anges nedan (nästa sida). Data finns även på filen `data1.m` som är en vanlig scriptfil. Använd valfri metod men for-satser eller andra repetitioner får *inte* användas. Programmet ska fungera även om man ändrar mätvärdena eller antalet punkter. Antalet (12) får inte stå utskrivet i klartext någonstans. Alla MATLABs funktioner är tillåtna att använda. Håll reda på själva var det ska vara rad- respektive kolumn-vektorer och transponera när det behövs.

Markera mätpunkterna med någon symbol och rita den anpassade linjen i samma bild. Definiera axlarnas `min` och `max` lite *utanför* mätområdet så att alla punkter ligger innanför bildrutan och ingen precis på kanten. Det ser snyggare ut och ger bättre överblick.

Beräkna också *rms-felet*. Definition finns nedan (nästa sida). Skriv ut värdet i bilden (funktionen *text*) eller i MATLAB-fönstret. For-sats får inte användas! Använd MATLABs inbyggda funktioner eller skriv ett enkelt uttryck där de aktuella vektorerna och matrisen används.

$$rms - fel = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(c_1 + c_2 x_i - y_i)^2}$$

$\mathbf{x} = \begin{bmatrix} 11 & 12 & 15 & 28 & 45 & 52 & 57 & 75 & 81 & 88 & 93 & 97 \end{bmatrix}$
$\mathbf{y} = \begin{bmatrix} 1.0 & 1.0 & 1.5 & 6.0 & 9.0 & 10.5 & 11.0 & 16.5 & 9.5 & 8.0 & 12.5 & 12.5 \end{bmatrix}$

Uppgift 2 – Vinterdag

På binärfilen *vinter.mat* finns mätdata som visar temperaturen under en stor del av ett vinterdygn. Den första kolumnen är timmen för tidmätningen och den andra visar minuterna. Första mätningen är gjord kl. 6.00, den andra 6.10 (tio minuter över 6) o.s.v. med jämnt steg ända till sista mätningen kl. 22.00. Skriv ett MATLAB-program som hämtar in (t.ex med *load*) data från filen och med hjälp av minstakvadratmetoden anpassar en kurva av följande typ till temperaturdata:

$$T = c_1 + c_2 sin(2\pi t/N) + c_3 cos(2\pi t/N)$$

Välj först N så att funktionernas period blir rimlig! t är tiden för mätning och T är temperaturen. Plotta mätpunkterna och rita upp den anpassade kurvan. Vad blir dygnets högsta temperatur enligt den anpassade kurvan?

Uppgift 3 – Cirkel genom tre punkter samt cirkelanpassning

Formeln $c_1 + c_2 x + c_3 y = x^2 + y^2$ är ett bekvämt sätt att ange cirkelns ekvation då man vill hitta den entydigt bestämda cirkel som passerar genom tre givna punkter. De tre punkterna ger tre ekvationer ur vilka man kan lösa ut c_1, c_2 och c_3. Formeln går också att använda för att från fler än tre punkter formulera ett överbestämt ekvationssystem som löses med minstakvadratmetoden och ger en cirkelapproximation av mätdata. För att rita den erhållna cirkeln bör man ha dess mittpunkt (x_c, y_c) och radie R (se MATLAB-labb 1!). x_c, y_c och R kan härledas från formeln ovan m.h.a. kvadratkomplettering eller genom att utgå från den vanliga ekvationen för cirkeln $(x - x_m)^2 + (y - y_m)^2 = R^2$, utvecka kvadraterna och identifiera de två uttrycken.

Här ska tre snarlika uppgifter lösas. Entydigt bestämda ekvationssystem och överbestämda system löses på samma sätt i MATLAB, med operatorn \, t.ex. `A\b`. Skriv en funktion som tar x- och y-koordinater som indata och beräknar cirkelparametrarna. Indatapunkterna ska plottas och beräknad cirkel ritas upp. Man kan välja att lägga ritandet i funktionen som beräknar cirkeln eller utanför funktionen. Om man väljer att rita utanför måste funktionen returnera cirkelparametrarna.

Krav: Samma funktion som beräknar cirkelparametrar ska användas i alla tre uppgifterna 4a, 4b och 4c.

4a Använd tre förutbestämda punkter. Låt funktionen beräkna cirkeln som passerar genom alla tre punkterna. Plotta punkterna och rita cirkeln.

4b Anpassa en cirkel till minst sex givna punkter. Här är förslag men välj andra om ni vill: (-2,2), (-1,5), (2,4), (-1,0), (1,0), (3,1). Använd samma funktion som i 4a.

4c Till slut, utnyttja grafisk inmatning med MATLAB - kommandot `ginput`, och mata in punkter som ges till cirkelanpassningsfunktionen.

Uppgift 4 – Rita med rekursion

Med rekursiva algoritmer kan man med kort programkod rita komplicerade och ofta vackra mönster. Här är uppgiften att rita en Sierpinskitriangel rekursivt. Se exempel på nästa sida! Algoritmförslag för rekursiv funktion, exklusive basfallet:

- Starta med en triangel. Kalla hörnpunkterna $X1$, $X2$ och $X3$.
- Beräkna mittpunkterna på triangelsidorna. Kalla dem $M1$, $M2$ och $M3$.
- Fyll triangelytan $M1 - M2 - M3$ med färg.
- Anropa funktionen på de tre trianglarna
  * $X1 - M1 - M3$
  * $M1 - X2 - M2$
  * $M3 - M2 - X3$

En rekursiv algoritm måste ha ett basfall som inte är rekursivt annars avslutas aldrig rekursionen. Här är det lämpligt att låta antalet rekursionsnivåer avgöra när rekursionen ska vända. Låt den rekursiva funktionen ha antalet nivåer, n, som parameter. Funktionens huvud (första rad) kan vara

```
function sierpinski(x,y,n)
```

x och y är vektorer med tre element vardera och utgör triangelns hörnpunkter. n är antalet rekursionsnivåer. I exemplen till höger på nästa sida finns Sierpinskitrianglar med $n = 2$, $n = 4$ och $n = 6$, uppifrån och ned. När funktionen anropar sig själv på de tre småtrianglarna (***** ovan) så används $n-1$ i alla tre anropen. När $n == 0$ så ska inga rekursiva anrop göras och rekursionen vänder. I bilden överst till höger är funktionen anropad med $n = 2$. Röda triangeln i mitten ritas först och de tre rekursiva anropen (enligt ***** ovan) ger de tre mindre röda trianglarna. Därefter vänder rekursionen och inget mer ritas.

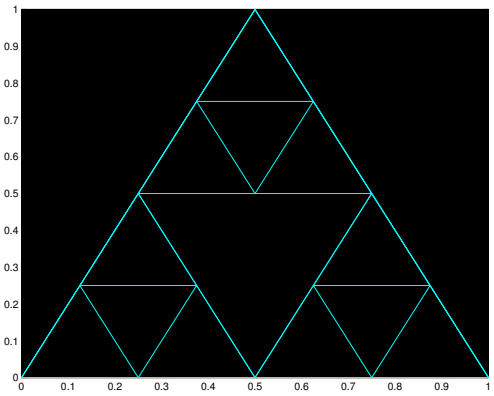
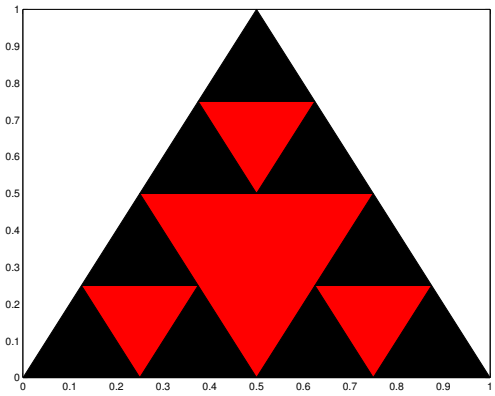
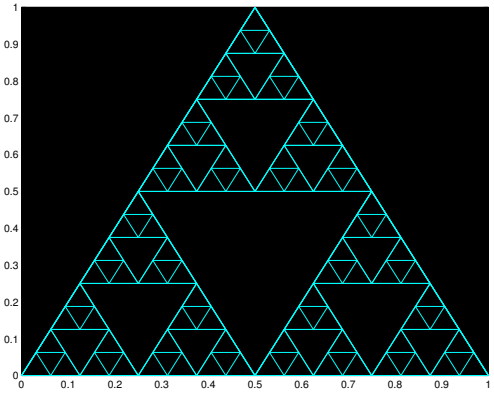
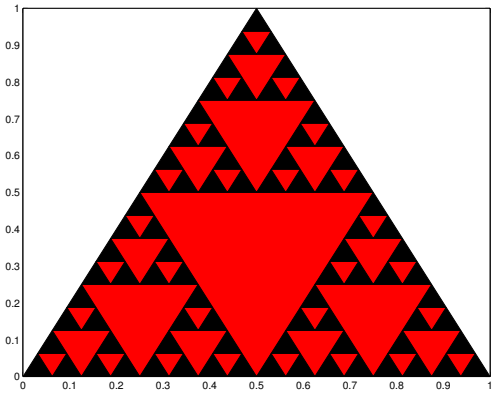
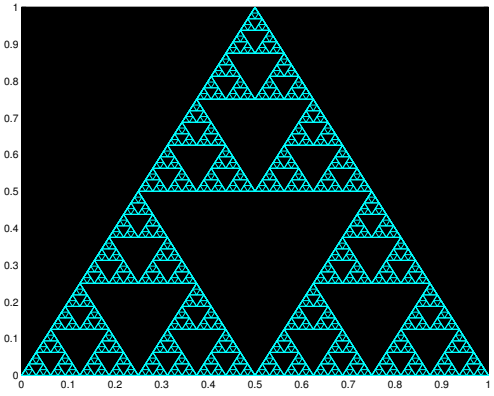
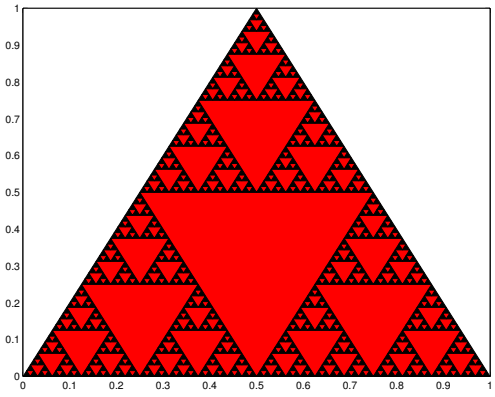
Den svarta bakgrundstriangeln eller bakgrunden är inte en del av rekursionen. Låt en annan funktion (som ej är rekursiv) rita bakgrund och anropa den rekursiva funktionen.

Lägg in en liten paus i den rekursiva funktionen så kan man se hur de rekursiva anropen bygger upp bilden. Använd MATLAB-funktionen som anropas t.ex. `pause(0.01)` Parametern till funktionen anger förstås paustiden. Experimentera fram lämpligt parametervärde.

Variation:

Om ni vill, gör istället något av följande:

- Rita streck istället för att fylla triangelytor med färg. Det gör inget om man ritar samma streck flera gånger! Se vänstra kolumnen i figuren på nästa sida.
- Välj en annan färgsättning, flera färger eller slumpvisa färger.
- Välj en annan definitition av Sierpinskitriangeln.
- Välj en helt annat rekursiv figur, bara den inte är mycket enklare än Sierpinskitriangeln. Några exempel är snöflinga och träd.

Uppgift 5 – Brownsk rörelse

Skriv MATLAB - kod som ritar spåren efter 4-10 stycken partiklar som rör sig (förflyttas) slumpmässigt i xy-planet. Definiera en storlek, `size`, och sätt axlarna till `[-size size -size size]`. Låt alla starta i mitten av fönstret och låt varje förflyttning styras av två slumptal, ett tal för vardera x-led och y-led. Det går bra att direkt använda de tal man får genom att anropa MATLAB - funktionen `randn` som ger normalfördelade slumptal, alltså både positiva och negativa värden. `randn` kan ges heltalsparametrar. En parameter ger kvadratisk matris med slumptal i alla positioner, två parametrar `randn(n,m)` ger $n \times m$–matris . Observera att slumptalen används för att definiera en liten förflyttning (dx, dy) och inte den nya positionen. I en repetition (for-sats) med minst ett par hundra steg beräknas nya värden till alla positioner och den senaste förflyttningen ritas som ett streck. Använd MATLAB - funktionen `pause(s)` efter varje slumpflyttning så att man ser rörelsen. Experimentera fram lämpligt värde på fördröjning. De olika "partikelspåren" ska ha olika färg och färgerna kan lämpligen lagras i en vektor, t.ex. `col = 'rmcg'`. Det enda som ska lagras i programmet är positioner och färger, en partikel har ingen representation utöver detta och färgen används när förflyttnings- strecket ritas.

Krav: Koduppprepning får inte förekomma. Man ska kunna lägga till en partikel utan att lägga till några satser till programmet. Det ska räcka att utöka vektorer och matriser.

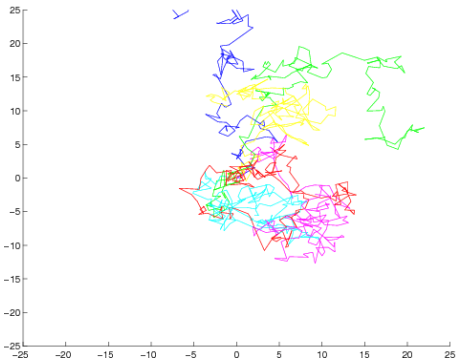
Uppgift 6 – Schackbräde

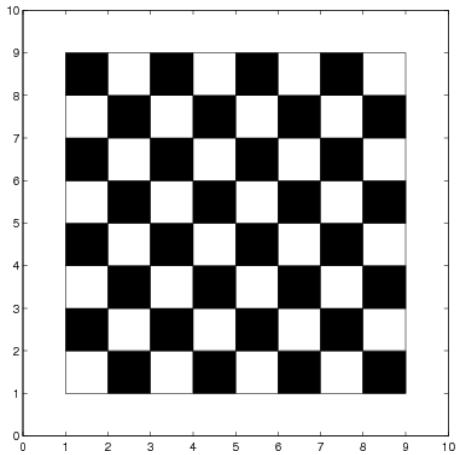
Skriv en MATLAB - funktion som ritar schackbrädesmönster ("rutig målarfärg"). Det duger bra att funktionen bara kan rita svart-vitt, att både längden och bredden är lika stora och att det är jämna tal.

Krav: Funktionen ska ha en parameter som anger storleken så att man ska kunna få godtyckligt (ev. endast jämnt) antal rutor ritade.

Det är trevligt men *inga krav* att man kan välja andra färger än svart och vitt, att man kan välja olika längd och bredd och att man välja även udda antal rutor.

Här nedan finns exempel på körningar av Brownsk rörelse och Schackbräde. Till vänster syns resultatet av att simulera 150 slumpflyttningar av sex partiklar. Färgerna som används är röd, magenta, cyan, blå, grön och gul.





Uppgift 7 – Rät linje med minstakvadratmetoden, vinkelräta avstånd

Den vanliga minstakvadratmetoden som används i uppgift 1 minimerar summan av kvadraterna på avstånden i *y-led* mellan mätpunkter och anpassad linje. Om man har samma enhet i x-led och y-led och vill att anpassningen ska bli likadan ifall man roterar sina mätpunkter (anpassad linje roteras på samma sätt som mätpunkterna) så är det bättre att istället minimera summan av kvadraterna på de *vinkelräta* avstånden mellan punkterna och linjen. Lösningen till detta minstakvadratproblem kan formuleras på flera sätt. Ett sätt är att använda egenvärden och egenvektorer till en matris som definieras enligt följande:

Vektorerna $\mathbf{x}$ och $\mathbf{y}$ är givna data. Börja med att beräkna deras respektive medelvärden x_m och y_m (`mean` i MATLAB). Bilda sedan de centrerade datavektorerna $\mathbf{x_c} = \mathbf{x} - x_m$ och $\mathbf{y_c} = \mathbf{y} - y_m$. Med hjälp av de centrerade vektorerna bildas den symmetriska 2×2-matrisen

$$\mathbf{A} = \begin{pmatrix} \mathbf{x_c}^{\mathrm{T}}\mathbf{x_c} & \mathbf{x_c}^{\mathrm{T}}\mathbf{y_c} \\ \mathbf{x_c}^{\mathrm{T}}\mathbf{y_c} & \mathbf{y_c}^{\mathrm{T}}\mathbf{y_c} \end{pmatrix}$$

Matrisen skapas *utan* for-satser.

Den bästa räta linjen enligt denna metod kommer att passera genom punkten (x_m, y_m). Linjens riktningsvektor utgörs av den egenvektor som svarar mot det *största* egenvärdet hos matrisen $\mathbf{A}$. Egenvektorerna är ortogonala (det vet man från algebrakursen, eller hur?) så egenvektorn som svarar mot det *minsta* egenvärdet är linjens normalvektor $\mathbf{n}$ och med hjälp av denna kan linjens ekvation skrivas

$$n_x(x - x_m) + n_y(y - y_m) = 0$$

Matlabkommandot *eig* ger två utdata om man vill:

`[V,lam] = eig(A)` ger egenvektorerna till `A` i matrisen `V` och egenvärdena i stigande ordning i diagonalen i matrisen `lam`. Egenvektor och egenvärde hör självklart ihop kolumnvis.

Anpassa en linje till givna data m.h.a. tekniken ovan. Plotta givna data med en symbol samt rita den anpassade linjen. Liksom i uppgift 1 ska ni se till att bilden omfattar ett större område än det som definieras av största och minsta värden hos mätpunkterna.
Data till uppgiften finns på scriptfilen `data2.m`.